

# VU Research Portal

## Safe and automatic live update

Giuffrida, C.

2014

### **document version**

Publisher's PDF, also known as Version of record

[Link to publication in VU Research Portal](#)

### **citation for published version (APA)**

Giuffrida, C. (2014). *Safe and automatic live update*. [PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam].

### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

### **Take down policy**

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

### **E-mail address:**

[vuresearchportal.ub@vu.nl](mailto:vuresearchportal.ub@vu.nl)

# Contents

<b>Acknowledgements</b>	<b>vii</b>
<b>Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xvii</b>
<b>Publications</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Safe and Automatic Live Update for Operating Systems</b>	<b>9</b>
2.1 Introduction . . . . .	10
2.1.1 Contribution . . . . .	12
2.2 Background . . . . .	12
2.2.1 Safe Update State . . . . .	12
2.2.2 State Transfer . . . . .	13
2.2.3 Stability of the update process . . . . .	15
2.3 Overview . . . . .	16
2.3.1 Architecture . . . . .	16
2.3.2 Update example . . . . .	17
2.3.3 Limitations . . . . .	18
2.4 Live Update Support . . . . .	19
2.4.1 Programming model . . . . .	19
2.4.2 Virtual IPC endpoints . . . . .	20
2.4.3 State filters . . . . .	20

2.4.4	Interface filters . . . . .	21
2.4.5	Multicomponent updates . . . . .	21
2.4.6	Hot rollback . . . . .	22
2.5	State Management . . . . .	22
2.5.1	State transfer . . . . .	23
2.5.2	Metadata instrumentation . . . . .	24
2.5.3	Pointer transfer . . . . .	25
2.5.4	Transfer strategy . . . . .	26
2.5.5	State checking . . . . .	27
2.6	Evaluation . . . . .	28
2.6.1	Experience . . . . .	28
2.6.2	Performance . . . . .	31
2.6.3	Service disruption . . . . .	34
2.6.4	Memory footprint . . . . .	34
2.7	Related work . . . . .	35
2.8	Conclusion . . . . .	37
2.9	Acknowledgments . . . . .	37

### 3 Enhanced Operating System Security Through Efficient and Fine-grained

	<b>Address Space Randomization</b>	<b>39</b>
3.1	Introduction . . . . .	40
3.1.1	Contributions . . . . .	41
3.2	Background . . . . .	41
3.2.1	Attacks on code pointers . . . . .	41
3.2.2	Attacks on data pointers . . . . .	42
3.2.3	Attacks on nonpointer data . . . . .	42
3.3	Challenges in OS-level ASR . . . . .	42
3.3.1	$W \oplus X$ . . . . .	42
3.3.2	Instrumentation . . . . .	43
3.3.3	Run-time constraints . . . . .	43
3.3.4	Attack model . . . . .	43
3.3.5	Information leakage . . . . .	44
3.3.6	Brute forcing . . . . .	44
3.4	A design for OS-level ASR . . . . .	45
3.5	ASR transformations . . . . .	47
3.5.1	Code randomization . . . . .	48
3.5.2	Static data randomization . . . . .	49
3.5.3	Stack randomization . . . . .	51
3.5.4	Dynamic data randomization . . . . .	52
3.5.5	Kernel modules randomization . . . . .	52
3.6	Live rerandomization . . . . .	53
3.6.1	Metadata transformation . . . . .	53
3.6.2	The rerandomization process . . . . .	54

3.6.3	State migration . . . . .	55
3.6.4	Pointer migration . . . . .	56
3.7	Evaluation . . . . .	57
3.7.1	Performance . . . . .	57
3.7.2	Memory usage . . . . .	60
3.7.3	Effectiveness . . . . .	60
3.8	Related work . . . . .	63
3.8.1	Randomization . . . . .	63
3.8.2	Operating system defenses . . . . .	64
3.8.3	Live rerandomization . . . . .	64
3.9	Conclusion . . . . .	65
3.10	Acknowledgments . . . . .	65
<b>4</b>	<b>Practical Automated Vulnerability Monitoring Using Program State In-</b>	
	<b>variants</b>	<b>67</b>
4.1	Introduction . . . . .	68
4.2	Program State Invariants . . . . .	69
4.3	Architecture . . . . .	70
4.3.1	Static Instrumentation . . . . .	71
4.3.2	Indexing pointer casts . . . . .	72
4.3.3	Indexing value sets . . . . .	72
4.3.4	Memory management instrumentation . . . . .	73
4.3.5	Metadata Framework . . . . .	73
4.3.6	Dynamic Instrumentation . . . . .	74
4.3.7	Run-time Analyzer . . . . .	75
4.3.8	State introspection . . . . .	76
4.3.9	Invariants analysis . . . . .	76
4.3.10	Recording . . . . .	77
4.3.11	Reporting . . . . .	77
4.3.12	Feedback generation . . . . .	78
4.3.13	Debugging . . . . .	78
4.4	Memory Errors Detected . . . . .	78
4.4.1	Dangling pointers . . . . .	78
4.4.2	Off-by-one pointers . . . . .	78
4.4.3	Overflows/underflows . . . . .	79
4.4.4	Double and invalid frees . . . . .	79
4.4.5	Uninitialized reads . . . . .	79
4.5	Evaluation . . . . .	79
4.5.1	Performance . . . . .	80
4.5.2	Detection Accuracy . . . . .	83
4.5.3	Effectiveness . . . . .	84
4.6	Limitations . . . . .	85
4.7	Related Work . . . . .	86

4.8	Conclusion . . . . .	87
4.9	Acknowledgments . . . . .	88
<b>5</b>	<b>EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments</b>	<b>89</b>
5.1	Introduction . . . . .	90
5.2	Background . . . . .	91
5.3	System Overview . . . . .	93
5.4	Execution-driven Fault Injection . . . . .	95
5.5	Static Fault Model . . . . .	97
5.6	Dynamic Fault Model . . . . .	100
5.7	Evaluation . . . . .	103
5.7.1	Performance . . . . .	104
5.7.2	Memory usage . . . . .	105
5.7.3	Precision . . . . .	106
5.7.4	Controllability . . . . .	108
5.8	Conclusion . . . . .	110
5.9	Acknowledgments . . . . .	110
<b>6</b>	<b>Back to the Future: Fault-tolerant Live Update with Time-traveling State Transfer</b>	<b>111</b>
6.1	Introduction . . . . .	112
6.2	The State Transfer Problem . . . . .	114
6.3	System Overview . . . . .	116
6.4	Time-traveling State Transfer . . . . .	118
6.4.1	Fault model . . . . .	118
6.4.2	State validation surface . . . . .	119
6.4.3	Blackbox validation . . . . .	121
6.4.4	State transfer interface . . . . .	122
6.5	State Transfer Framework . . . . .	122
6.5.1	Overview . . . . .	123
6.5.2	State transfer strategy . . . . .	124
6.5.3	Shared libraries . . . . .	125
6.5.4	Error detection . . . . .	126
6.6	Evaluation . . . . .	126
6.6.1	Performance . . . . .	127
6.6.2	Memory usage . . . . .	129
6.6.3	RCB size . . . . .	129
6.6.4	Fault tolerance . . . . .	130
6.6.5	Engineering effort . . . . .	133
6.7	Related Work . . . . .	134
6.7.1	Live update systems . . . . .	134
6.7.2	Live update safety . . . . .	134

6.7.3	Update testing . . . . .	135
6.8	Conclusion . . . . .	135
6.9	Acknowledgments . . . . .	136
<b>7</b>	<b>Mutable Checkpoint-Restart: Automating Live Update for Generic Long-running C Programs</b>	<b>137</b>
7.1	Introduction . . . . .	138
7.2	Background and Related Work . . . . .	139
7.2.1	Quiescence detection . . . . .	139
7.2.2	Control migration . . . . .	140
7.2.3	State transfer . . . . .	140
7.3	Overview . . . . .	140
7.4	Profile-guided Quiescence Detection . . . . .	142
7.4.1	Quiescent points . . . . .	142
7.4.2	Instrumentation . . . . .	143
7.4.3	Quiescence detection . . . . .	144
7.5	State-driven Mutable Record-replay . . . . .	146
7.5.1	Control migration . . . . .	147
7.5.2	Mapping operations . . . . .	147
7.5.3	Immutable state objects . . . . .	148
7.6	Mutable GC-style Tracing . . . . .	150
7.6.1	Mapping program state . . . . .	150
7.6.2	Precise GC-style tracing . . . . .	152
7.6.3	Conservative GC-style tracing . . . . .	153
7.7	Violating Assumptions . . . . .	154
7.8	Evaluation . . . . .	155
7.8.1	Engineering effort . . . . .	155
7.8.2	Performance . . . . .	158
7.8.3	Update time . . . . .	160
7.8.4	Memory usage . . . . .	162
7.9	Conclusion . . . . .	163
7.10	Acknowledgments . . . . .	163
<b>8</b>	<b>Conclusion</b>	<b>165</b>
	<b>References</b>	<b>171</b>
	<b>Summary</b>	<b>197</b>
	<b>Samenvatting</b>	<b>199</b>